Linear Combination of Transformations

Marc Alexa

Interactive Graphics Systems Group, Technische Universität Darmstadt [*]

Abstract

Geometric transformations are most commonly represented as square matrices in computer graphics. Following simple geometric arguments we derive a natural and geometrically meaningful definition of scalar multiples and a commutative addition of transformations based on the matrix representation, given that the matrices have no negative real eigenvalues. Together, these operations allow the linear combination of transformations. This provides the ability to create weighted combination of transformations, interpolate between transformations, and to construct or use arbitrary transformations in a structure similar to a basis of a vector space. These basic techniques are useful for synthesis and analysis of motions or animations. Animations through a set of key transformations are generated using standard techniques such as subdivision curves. For analysis and progressive compression a PCA can be applied to sequences of transformations. We describe an implementation of the techniques that enables an easy-to-use and transparent way of dealing with geometric transformations in graphics software. We compare and relate our approach to other techniques such as matrix decomposition and quaternion interpolation.

CR Categories: G.1.1 [Numerical Analysis]: Interpolation—Spline and piecewise polynomial interpolation; I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric Transformations; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation;

Keywords: transformations, linear space, matrix exponential and logarithm, exponential map

1 Introduction

Geometric transformations are a fundamental concept of computer graphics. Transformations are typically represented as square real matrices and are applied by multiplying the matrix with a coordinate vector. Homogeneous coordinates help to represent additive transformations (translations) and multiplicative transformations (rotation, scaling, and shearing) as matrix multiplications. This representation is especially advantageous when several transformations have to be composed: Since the matrix product is associative all transformation matrices are multiplied and the concatenation of the transformations is represented as a single matrix.

[*]email:alexa@gris.informatik.tu-darmstadt.de

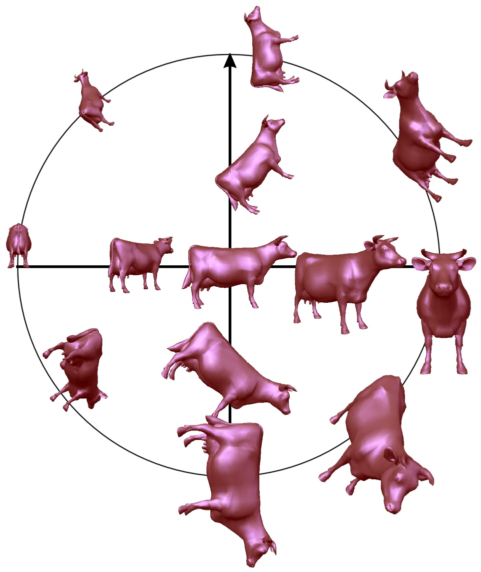

Figure 1: A two-dimensional cow space: Two transformations A and B, both of which include a rotation, a uniform scale, and a translation, form a two-dimensional space of transformations. In this space $(0,0)$ is the identical transformation, $(1,0)$ and $(0,1)$ represent the specified transformations A and B.

For the representation of motion it is necessary to interpolate from one given transformation to another. The common way in computer graphics for blending or interpolating transformations is due to the pioneering work of Shoemake [Shoemake 1985; Shoemake 1991; Shoemake and Duff 1992]. The approach is to decompose the matrices into rotation and stretch using the polar decomposition and then representing the rotation using quaternions. Quaternions are interpolated using SLERP and the stretch matrix might be interpolated in matrix space. Note, however, that the quaternion approach has drawbacks. We would expect that "half" of a transformation T applied twice would yield T. Yet this is not the case in general because the factorization uses the matrix product, which is not commutative. In addition, this factorization induces an order dependence when handling more than two transformations.

Barr et al. [1992], following Gabriel & Kajiya [1985], have formulated a definition of splines using variational techniques. This allows one to satisfy additional constraints on the curve. Later, Ramamoorthi & Barr [1997] have drastically improved the computational efficiency of the technique by fitting polynomials on the unit quaternion sphere. Kim et al. [1995] provide a general framework for unit quaternion splines. However, compared to the rich tool-box for splines in the euclidean space, quaternions splines are still difficult to compute, both in terms of programming effort as well as in terms of computational effort.

We identify as the main problem of matrix or quaternion representations that the standard operators are not commutative. In this work we will give geometrically meaningful definitions for scalar product and addition of transformations based on the matrix representation. We motivate the definitions geometrically. The defini-

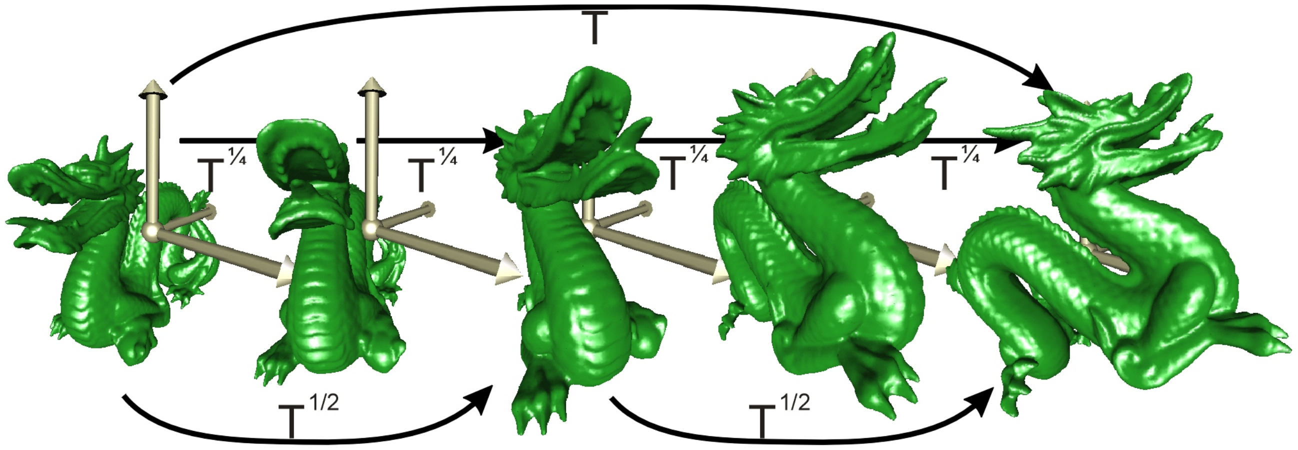

Figure 2: Defining scalar multiples of transformations. Intuitively, “half” of a given transformation T should be so defined that applying it twice yields T. This behavior is expected for arbitrary parts of transformations. Consequently, scalar multiples are defined as powers of the transformation matrices.

tions lead to the use of an exponential map into the Lie group of geometric transformations. Once this connection is established we compare our definition to other approaches. The implementation of this approach uses a transform object that transparently offers scalar product and addition operators. This gives API users an easy-to-use, intuitive, and flexible tool whenever it is desirable to combine transforms rather than composing them.

2 Related work

Our approach essentially uses interpolation in Lie groups by means of the exponential map [Marthinsen 2000]. Grassia has introduced this idea for 3D graphics to represent the group of rotations [Grassia 1998]. The group of rotations SO(3) and the group of rigid body motions SE(3) are commonly used for motion planning in the field of robotics. Park and Ravani compute interpolating splines for a set of rotations in SO(3) [Park and Ravani 1997]. They compare the groups SO(3) and SU(2) (the group of unit quaternions) in detail. One main advantage of using SO(3) for interpolation is bi-invariance, e.g. if two sets of rotations are connected with an affine mapping the resulting curves are connected by the same map. In our context, this property is naturally contained as part of linearity. Zefran analyzes SE(3) for general problems in motion planning (see [Zefran 1996] and the references therein). The main problem is that the respective spaces have non-Euclidean geometry and one has a choice of several reasonable metrics [do Carmo 1992; Zefran et al. 1996]. Once a metric is defined, variational methods are used to determine an interpolant [Zefran and Kumar 1998]. In our approach we have rather traded the problem of defining the geometrically most meaningful metric and solving a variational problem for simplicity, ease-of-use and transparency. In addition, we extend these methods from rotations and rigid body motion to general transformations.

The results of our techniques are on an abstract level identical to those from Geometric Algebra (GA) [Hestenes 1991], a field recently introduced to the graphics community [Naeve and Rockwood 2001]. Current implementations of GA [Dorst and Mann 2001] use explicit representations of all sub-elements (i.e. points, lines, planes, volumes), which results in $\mathbb{R}^3$ being represented with 8×8 matrices. In a sense, our approach could be seen as an alternative implementation using more complex operations on the matrices, however, in smaller dimension.

3 Motivation and definition of scalar multiples of transformations

Suppose that we have some transformation, T, and we want to define a scalar mutliple, $\alpha \odot T$. What conditions should such a scalar multiple satisfy? Well, in the particular case $\alpha = \frac{1}{2}$, i.e., ”half” of T, we want the resulting transformation to have the property that when it’s applied twice, the result is the original transformation T, i.e., that

$$\left(\frac{1}{2}\odot T\right)\circ\left(\frac{1}{2}\odot T\right) = T; \tag{1}$$

an illustration of our goal is given in Figure 2.

We’ll require analogous behavior for one-third of a transformation, one fourth, and so forth. We’ll also want $\alpha \odot T$ to be a continuous function of both α and T.

Let’s explore what this entails by examining the consequences for some standard transformations: translation, rotation, and scaling.

Translation: If T is a translation by some amount $\mathbf{v}$, then clearly translation by $\alpha\mathbf{v}$ is a good candidate for $\alpha \odot T$; it satisfies the requirements of equation 1 and its analogues, and has the advantage that it’s also a translation.

Rotation: If T is a rotation of angle θ about the axis $\mathbf{v}$, then rotation about the axis $\mathbf{v}$ by angle $\alpha\theta$ is a good candidate for $\alpha \odot T$, for simialr reasons.

Scaling: Finally, if T is a scaling transformation represented by a scale-matrix with diagonal entries $d_1, d_2, \dots$ then $diag(d_1^\alpha, d_2^\alpha, \dots)$ is a candidate for $\alpha \odot T$.

In all three cases, we see that for positive integer values of α, our candidate for $\alpha \odot T$ corresponds to T^α; the same is true for the matrix representing the transformation. If we had a way to define arbitrary real powers of a matrix, we’d have a general solution to the problem of defining scalar multiples; we’d define $\alpha \odot T$ to be T^α (where what we mean by this is that $\alpha \odot T$ is the transformation representated by the matrix M^α, where M is the matrix for T).

Fortunately, for a very wide class of matrices (those with no negative real eigenvalues), there is a consistent definition of M^α, and computing M^α is not particularly difficult (see Appendices A and C). Furthermore, it has various familiar properties, the most critical being that $M^\alpha M^\beta = M^{\alpha+\beta} = M^\beta M^\alpha$ (i.e. scalar multiples of the same transform commute), and $M^0 = I$ (the identity matrix). Some other properties of exponents do *not* carry over from real-number arithmetic, though: in general it's not true that $(AB)^\alpha = A^\alpha B^\alpha$. One more property is important: although a matrix may have two (or more) square roots (for example, both the identity and the negative identity are square roots of the identity!), for matrices with non negative-real eigenvalues, one can define a preferred choice of M^α which is continuous in M and α.

While techniques for computing parts of the above transformations are well known (see e.g. [Shoemake 1985; Shoemake 1991; Shoemake and Duff 1992; Park and Ravani 1997]) the idea of our approach is that taking powers of transformation matrices works for arbitrary transformations without first factoring the matrix into these components.

Following this intuitive definition of scalar multiples of transformations we need a commutative addition for transformations. Together, these operations will form the basic building blocks for linear combination of transformation.

4 Commutative addition of transformations

In this section, we motivate and define an operation we'll call "addition of transformations" – the word "addition" meant to remind the reader that the operation being defined is *commutative*. The ordinary matrix product combines two matrices by multiplying one by the other, which is not symmetric in the factors. For a commutative operation we rather expect the two transformations to be applied at the same time, or intertwined. We want to stress that the addition is not intended to replace the standard matrix product but to complement it. Clearly, both will have their uses and one has to choose depending on the effect to be achieved.

Let A, B be two square real matrices of the same dimension. Clearly, AB and BA are different in general, however, are the same if A and B commute. In this case the standard matrix product is exactly what we want, in all other cases we need to modify the product operation. The main idea of this work is to break each of the transformations A and B into smaller parts and perform (i.e. multiply) these smaller parts alternately.

Small parts of A and B are generated by scalar multiplication with a small rational number, e.g. $1/n$. Loosely speaking, we expect that $\left(A^{1/n}B^{1/n}\right)^n$ differs less from $\left(B^{1/n}A^{1/n}\right)^n$ than AB from BA. This is because a large part of the product is the same and the difference is represented by $n^{-1} \odot A$ respectively $n^{-1} \odot B$. Since $0 \odot X = I$ this difference would vanish for $n^{-1} \Rightarrow 0$ and we consequently define

$$A \oplus B = \lim_{n\to\infty} \left(A^{\frac{1}{n}} B^{\frac{1}{n}}\right)^n. \tag{2}$$

The idea of this definition is visualized in Figure 3. Several questions arise:

Existence Does the limit exist? Does it exist for all inputs? Is it real if the input is real?

Commutativity Is the addition indeed commutative?

Geometric properties What geometric properties has the new definition? For example, is the addition of two rotations a rotation?

The questions regarding existence and commutativity of the two operations are discussed in Appendix B. It can be shown that the limit indeed exists under reasonable conditions. Here we analyze some geometric properties of the addition.

The addition was designed to be commutative while preserving the properties of the standard matrix product. Thus, it is desirable that $A \oplus B = AB$ if $AB = BA$. If A and B commute then

$$A^n = \left(BAB^{-1}\right)^n = BAB^{-1}BAB^{-1}B\cdots B^{-1} = BA^nB^{-1},$$

i.e. also A^n and B commute. The same argument leads to $A^nB^n = B^nA^n$ and, assuming again that primary roots exist and are continous in their inputs, this result extends also to $A^{1/n}B^{1/n} = B^{1/n}A^{1/n}$. Thus

$$AB = \left(A^{\frac{1}{n}}\right)^n \left(B^{\frac{1}{n}}\right)^n = \left(A^{\frac{1}{n}} B^{\frac{1}{n}}\right)^n$$

and assuming the limit for $n \to \infty$ exists it follows that the matrix product and $\oplus$ are indeed the same if A and B commute. Furthermore, since A commutes with A^{-1} the inverse of $\oplus$ is the standard matrix (product) inverse.

Another important geometric property is the measure (area, volume) of a model. The change of this measure due to a transformation is available as the determinant of the matrix. Note that the order of two transformations is irrelevant for the change in size, i.e. $\det(AB) = \det(A)\det(B) = \det(B)\det(A) = \det(BA)$. It is easy to see that the addition of transformations conforms with this invariant:

$$\begin{aligned}\det(A \oplus B) &= \det\left(\lim_{n\to\infty}\left(A^{1/n}B^{1/n}\right)^n\right)\\ &= \det\left(\lim_{n\to\infty}\left(A^{1/n}\right)^n\right)\det\left(\lim_{n\to\infty}\left(B^{1/n}\right)^n\right)\\ &= \det(A)\det(B).\end{aligned}$$

In conclusion, the geometric behavior of $\oplus$ is very similar to the standard matrix product. Loosely speaking, $A \oplus$B is the application of A and B at the same time.

5 Computation and Implementation

Both the addition and scalar multiplication operators can be computed using matrix exponential and logarithm (see Appendix A for details). The definition of the matrix exponential is analogous to the scalar case, i.e.

$$e^A = \sum_{k=0}^{\infty} \frac{A^k}{k!}, \tag{3}$$

which immediately defines the matrix logarithm as its inverse function:

$$e^X = A \iff X = \log A. \tag{4}$$

The existence of matrix logarithms (as well as matrix roots) is discussed in Appendix B. Here, it may suffice to say that logarithms exist for transformation matrices, given that the transformation contains no reflection.

Using exponential and logarithm scalar multiples may be expressed as

$$r \odot A = e^{r \log A} \tag{5}$$

and the limit in Equation 2 is equivalent to

$$A \oplus B = e^{\log A + \log B}. \tag{6}$$

Using these equations a linear combination of an arbitrary number of transformations T_i with weights w_i is computed as

$$\boxed{\bigoplus_i w_i \odot T_i = e^{\sum_i w_i \cdot \log T_i}} \tag{7}$$

Note that the use of the exponential and logarithm hint at potential problems of this approach, or more generally, show the the

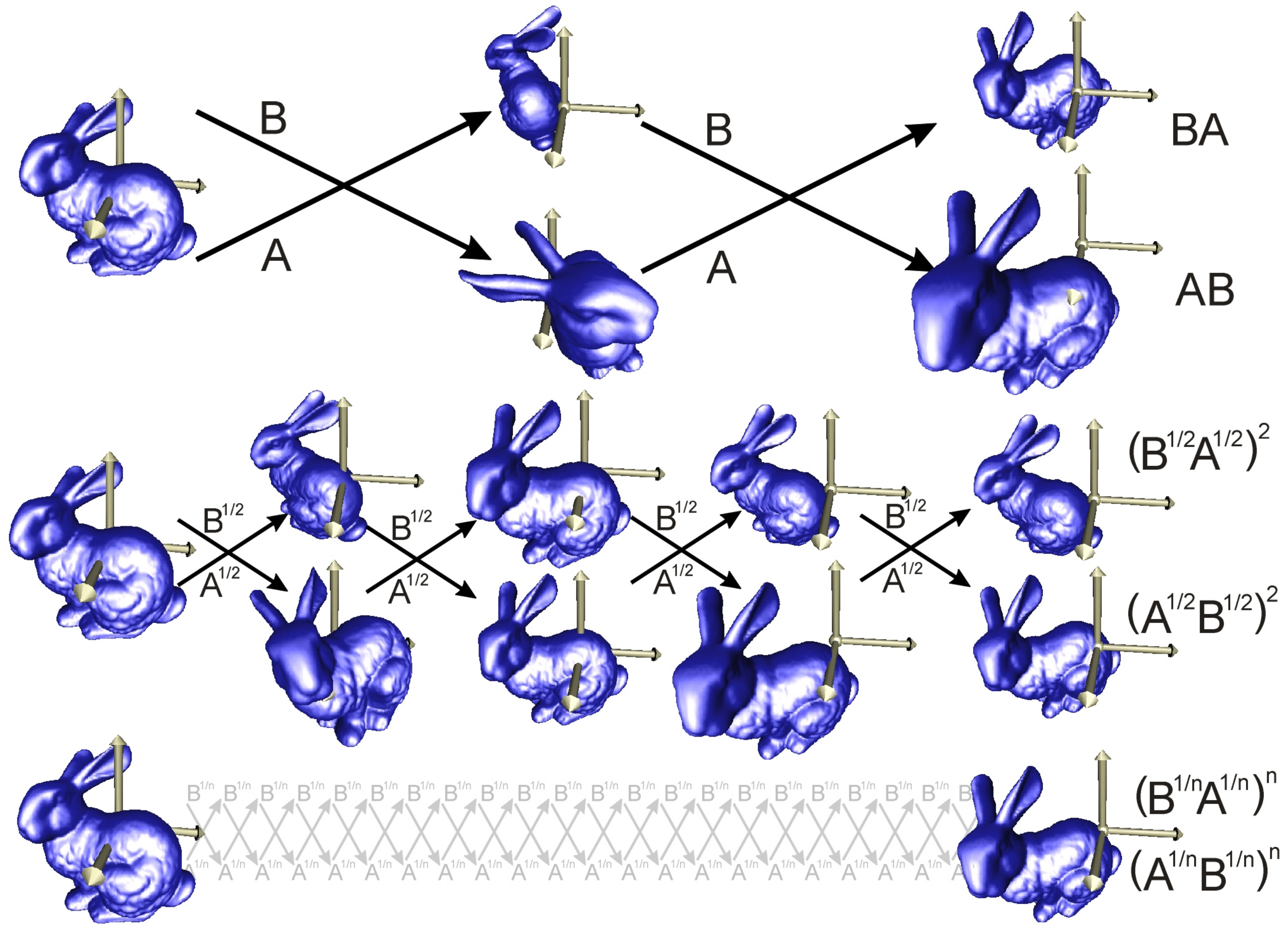

Figure 3: The addition of transformations. Given two transformations, A and B, applying one after the other (i.e. multiplying the matrices) generally leads to different results depeneding on the order of the operations. By performing n-th parts of the two transformations in turns the difference of the two orders becomes smaller. The limit $n \to \infty$ could be understood as performing both transformations concurrently. This is the intuitive geometric definition of a commutative addition for transformations based on the matrices.

non-linearity and discontinuity between the group of transformations and the space in which we perform our computations (i.e. the corresponding algebra). For example, both operators are in general not continous in their input, i.e. small changes in one of the transformations might introduce large changes in the result. Further potential problems and limitations are discussed together with applications in Section 6.

In order to implement this approach, routines for computing matrix exponential and logarithm are required. We suggest the methods described in Appendix C because they are stable and the most complex operation they require is matrix inversion, making them easy to integrate in any existing matrix package.

Using an object-oriented programming language with operator overloading it is possible to design a transform object that directly supports the new operations. The important observation is that the logarithm of a matrix has to be computed only once at the instantiation of an object. Any subsequent operation is performed in the log-matrix representation of the transformation. Only when the transformation has to be sent to the graphics hardware a conversion to original representation (i.e. exponentiation) is necessary.

Our current implementation needs $3 \cdot 10^{-5}$sec to construct a transform object, which is essentially the time needed to compute the matrix logarithm. The conversion to standard matrix representation (i.e. exponentiation) requires $3 \cdot 10^{-6}$sec. Timings have been acquired on a 1GHz Athlon PC under normal working conditions. Note that for most applications transform objects are created at the initialization of processes, while the conversion to standard representation is typically needed in in every frame. However, we have found the 3μs necessary for this conversion to be negligible in practice.

6 Applications & Results

Using the implementation discussed above, several interesting applications are straightforward to implement.

6.1 Smooth animations

A simple animation from a transformation state represented by A to a transformation B is achieved with $C(t) = (1-t) \odot A \oplus t \odot B, t \in [0,1]$. Using a cubic Bezier curve [Hoschek and Lasser 1993] allows one to define tangents in the start and endpoint of the interpolation. Using the Bezier representation, tangents are simply defined by supplying two transformations. Tangents could be used to generate e.g. fade-in/fade-out effects for the transformation. Figure 4 shows a linear and cubic interpolation of two given transformations.

To generate a smooth interpolant through a number of key frame transformations T_i one can use standard techniques from linear spaces such as splines [Bartels et al. 1985] or subdivision curves

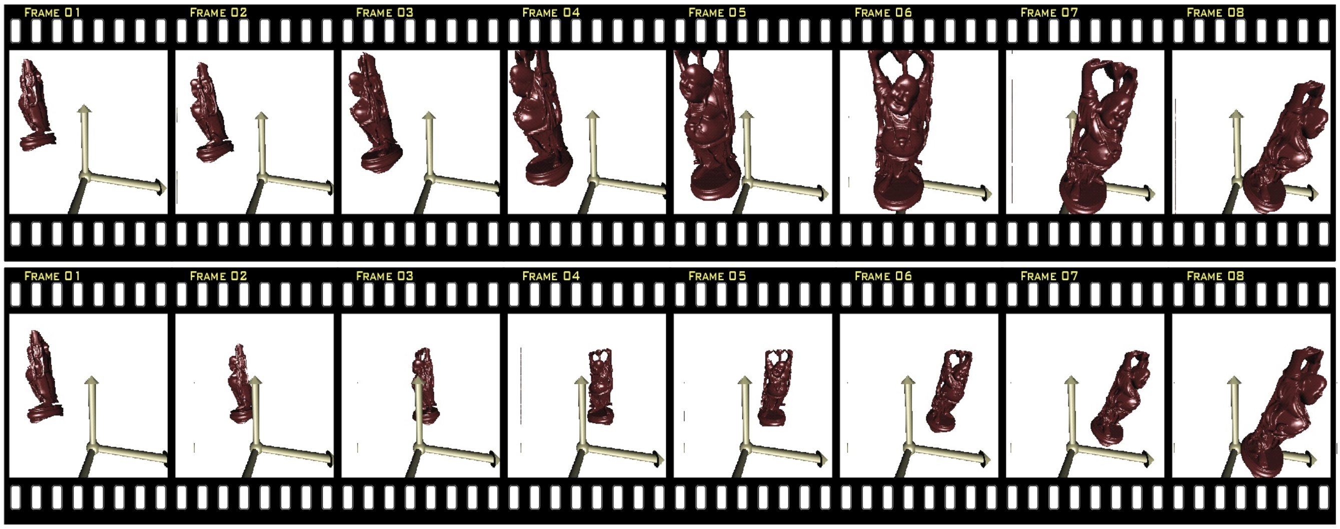

Figure 4: Interpolation sequences between given transformations A and B. The top row shows a simple linear interpolation using the matrix operators defined here, i.e. $(1-t)\odot A \oplus t \odot B$. The bottom row shows a Bezier curve from A to B with additional control transformations. These extra transformations define the tangents in the start and end point of the sequence.

[Zorin and Schröder 1999]. Note that the transparent implementation of the operators allows solving linear systems of equations in transformations using standard linear algebra packages. Using these techniques one can solve for the necessary tangent matrices which define e.g. a cubic spline. However, we find an interpolating subdivision scheme (e.g. the 4pt scheme [Dyn et al. 1987]) particularly appealing because it is simple to implement.

It seems that implementations of quaternion splines or other elaborated techniques are hardly available in common graphics APIs. Note how simple the implementation of interpolating or approximating transformation curves is with the approach presented here. One simply plugs the transform object into existing implementations for splines in Euclidean spaces.

The exponential map, on the other hand, has some drawbacks. Essentially, a straight line in parameter space doesn't necessarily map to a straight line (i.e. a geodesic) in the space of transformations. This means the linear interpolation between two transformations as defined above could have non-constant speed. Furthermore, also spline curves, which could be thought of as approximating straight lines as much as possible, are minimizers of a geometrically doubtful quantity. Nevertheless, we found the results pleasing.

We would also like to point at an interesting difference to quaternions: The log-matrix representation allows angles of arbitrary degree. Computing the logarithm of a rotation by π and then multiplying this log-matrix leads to a representation of rotations more than 2π. While this could be useful in some applications it might be disadvantageous in others. For example, the interpolation between two rotations of $\pm(\pi-\varepsilon)$ results in a rotation by almost 2π rather than a rotation by 2ε. However, using the tools presented in the following section this could be easily avoided.

We have compared the computation times of this approach with standard techniques. A SLERP based on quaternions between two rotation matrices is about 10 times faster than our approach. However, this is only true for the linear interpolation between two transformations. Quaternion splines are subtantially slower. They typically do not allow interactively adjusting the key transformations.

6.2 Factoring transformations

Transformations form a linear space in the log-matrix representation. This allows us to write any transformation as a kind of "linear combination" of transformations from an arbitrary "basis". The quotation marks indicate that this "linear combination" takes place in log-space – an associated space in which such combinations make sense. For example, three rotations R_x, R_y, R_z by an angle $0 < \phi < \pi$ around the canonical axes form a basis for the subspace of rotations. Since they are orthogonal, any transformation T can be factored by computing inner products of the log-representation:

$$x = \langle \log T, \log R_x \rangle, y = \langle \log T, \log R_y \rangle, z = \langle \log T, \log R_z \rangle, \qquad (8)$$

where the inner product is computed entry-wise, i.e. $\langle \{a_{ij}\}, \{b_{ij}\} \rangle = \sum a_{ij} b_{ij}$. Note that the values x, y, z do not represent Euler angles because the rotations around the axes are performed concurrently and not one after the other. Rather, x, y, z define axis and angle of rotation with $(x,y,z)/||(x,y,z)||$ being the axis and $(x+y+z)/\phi$ being the angle.

The factors x, y, z could be useful to avoid the interpolation problem mentioned at the end of the last Section. Assuming a representation as above the inner products will lead to $(x,y,z) \in [-r,r]^3$, where r depends on the angle of rotation in each of R_x, R_y, R_z. Specifically, values $-r$ and r represent the same orientation and one can imagine the angles to form a circle starting in $-r$ and ending in r with 0 diametrical to $\pm r$. To interpolate along the shortest path from R_1 to R_2 one chooses for each of the factors x_1, y_1, z_1 and x_2, y_2, z_2 the shorter path on the circle. Specifically, if the difference between two corresponding factors is larger than $|r|$, then the shorter interpolation path is via $\pm r$ rather than via 0.

Clearly, factoring has more applications than analyzing rotations. It could be done with respect to any (application specific) orthogonal or non-orthogonal basis. In order to find the representation of a transformation T in an arbitrary transformation basis $\{B_i\}$ we first compute the inner products of the bases. The matrix

$$V = \begin{pmatrix} \langle \log B_1, \log B_1 \rangle & \dots & \langle \log B_1, \log B_n \rangle \\ \vdots & \ddots & \vdots \\ \langle \log B_n, \log B_1 \rangle & \dots & \langle \log B_n, \log B_n \rangle \end{pmatrix}$$

describes a mapping from the orthogonal canonical base to the possible skew or deficient one formed by $\{B_i\}$. Computing the inverse

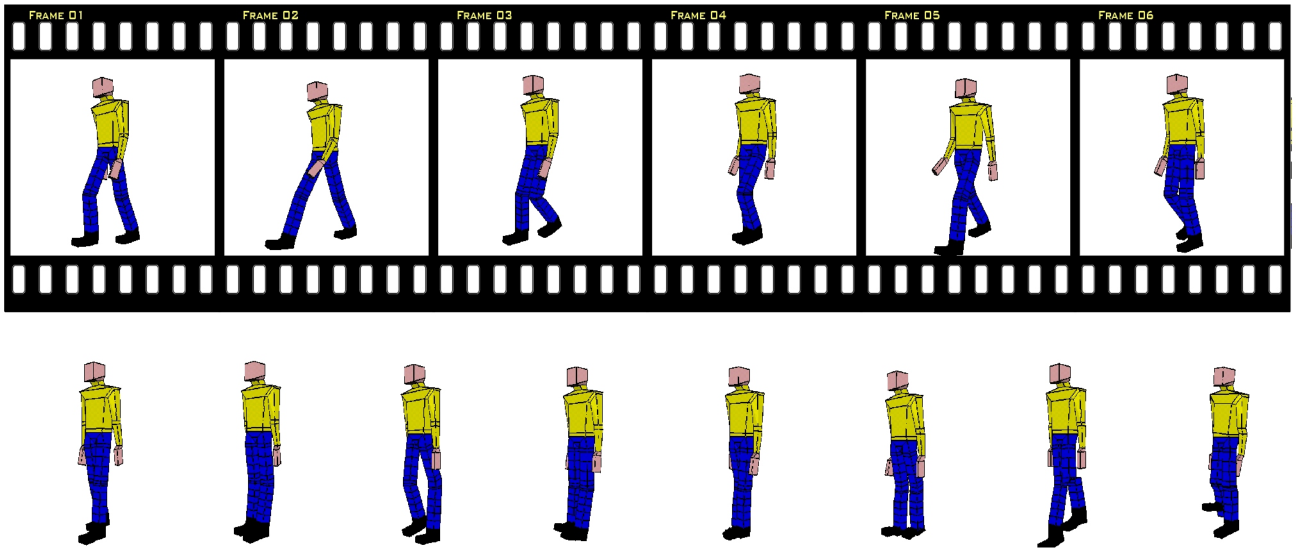

Figure 5: Animation analysis and compression based on the log-matrix representation. The upper rows shows 6 of 580 frames from a humanoid animation defined by key frame transformations in the joints of a skeleton. The respective log-matrices have been analyzed using the SVD. The bottom row shows the first 8 principal components.

of V reveals, first, whether the basis $\{B_i\}$ has full rank and, second, allows transforming a vector of inner products with the basis to the representation vector. We use singular value decomposition (SVD) [Golub and Van Loan 1989] for computing the inverse of V in order to get some information about the condition of the base.

Factoring has great applications in constraining transformations. The idea is to define an application-centered basis for a (sub)space of transformations and to factor and interpolate transformations in that (sub)space. Interpolating the factors allows one to generate smooth curves that naturally respect the constraints as defined by the subspace. In general, a suitable basis for the intended application-specific subspace might be hard to find. A simple solution is to first generate a number of permissible transformations T_i. The logarithms of the transformation matrices are written as rows of a matrix.

$$\begin{pmatrix} -\log T_0- \\ \vdots \\ -\log T_n- \end{pmatrix}$$

This matrix is decomposed using the SVD, which yields an orthonormal basis of the subspace of the permissible transformations.

6.3 Animation analysis

Analysis and compression of motions or animations is still a difficult subject. A reason might be that motions are typically non-linear so that powerful techniques such as a principal component analysis (PCA) [Jolliffe 1986] are difficult to apply. However, the techniques presented here allow the application of matrix techniques to analyze arbitrary transformations.

The SVD has been used by Alexa & Müller [2000] to generate a compact basis for a sequence of deforming meshes. The decomposition is applied to the vertex positions of key-frame meshes. This approach essentially decomposes the translational parts of an animation, while the rotational and scaling parts are not represented in a meaningful way. If an animation is mainly comprising local transformations a decomposition in the space of transformations would be more reasonable.

Using the linear matrix operators allows applying the SVD to sequences of transformations. As an example, we decompose a given skeleton animation of a walking humanoid. The matrices defining the local transformations are recorded over 580 key frames of the animation. The log-matrix representations of a key frame comprise a row of the representation matrix, which is then factored. The humanoid in the given animation follows the H-Anim specification [Web3D Consortium 1999] and has 17 joints, each of which provides 6 degrees of freedom (see Figure 5). The decomposition reveals that only 10 base elements are necessary to represent the 580 key frames faithfully. This reduces the originally $580 \cdot 17 \cdot 16 = 157760$ scalars to $580 \cdot 10 + 17 \cdot 16 \cdot 10 = 8520$, which is a compression factor of roughly 20.

This approach might also be applied to mesh animations. One has to assign a transformation to each primitive (e.g. vertex or face). This might require additional constraints as affine transformations offer more degrees of freedom than specified by single primitive. Applying a PCA to a deforming mesh could reveal an underlying process generating the deformation and, thus, essentially decompose it into affine transformations. This problem has been identified to be the key to efficient compression of animations [Lengyel 1999].

7 Conclusions

In this work, scalar multiples and commutative addition of transformations are defined, which are geometrically meaningful and easy to compute on the basis of the transformation matrices. Together, these operations enable the generation of linear combinations of transformations. This allows one to use common techniques for the synthesis and analysis of sets of transformations, e.g. to generate animations.

The main feature of this approach is the simplicity and flexibility in developing graphics software. We believe that many of the possible results of this approach might be generated by other means, though with considerably more programming effort and use of complex numerical techniques. We hope that the simple iterative implementations of matrix exponential and logarithm find their way in every graphics API.

Future work will concentrate on the aspect of analyzing motions and animations using linear techniques. Note that the approach works in any dimension and that we have not yet evaluated the resulting possibilities.

Acknowledgements

I would like to thank Roy Mathias for introducing me to matrix functions and Reinhard Klein for discussions on transformations and Lie groups. Johannes Behr has helped with coding and provided the implementation of the walking humanoid animation. Wolfgang Müller and the anonymous referees have given invaluable advice and helped tremendously in improving this document. This work has been supported by the BMBF grant "OpenSG PLUS".

References

ALEXA, M., AND MÜLLER, W. 2000. Representing animations by principal components. *Computer Graphics Forum 19*, 3 (August), 411–418. ISSN 1067-7055.

BARR, A. H., CURRIN, B., GABRIEL, S., AND HUGHES, J. F. 1992. Smooth interpolation of orientations with angular velocity constraints using quaternions. *Computer Graphics (Proceedings of SIGGRAPH 92) 26*, 2 (July), 313–320. ISBN 0-201-51585-7. Held in Chicago, Illinois.

BARTELS, R. H., BEATTY, J. C., AND BARSKY, B. A. 1985. An introduction to the use of splines in computer graphics.

DENMAN, E. D., AND BEAVERS JR., A. N. 1976. The matrix sign function and computations in systems. *Appl. Math. Comput. 2*, 63–94.

DO CARMO, M. P. 1992. *Riemannian Geometry*. Birkhäuser Verlag, Boston.

DORST, L., AND MANN, S. 2001. Geometric algebra: a computation framework for geometrical applications. *submitted to IEEE Computer Graphics & Applications*. available as http://www.cgl.uwaterloo.ca/~smann/Papers/CGA01.pdf.

DYN, N., LEVIN, D., AND GREGORY, J. 1987. A 4-point interpolatory subdivision scheme for curve design. *Computer Aided Geometric Design 4*, 4, 257–268.

GABRIEL, S. A., AND KAJIYA, J. T. 1985. Spline interpolation in curved space. In *SIGGRAPH '85 State of the Art in Image Synthesis seminar notes*.

GOLUB, G. H., AND VAN LOAN, C. F. 1989. *Matrix Computations*, second ed., vol. 3 of *Johns Hopkins Series in the Mathematical Sciences*. The Johns Hopkins University Press, Baltimore, MD, USA. Second edition.

GRASSIA, F. S. 1998. Practical parameterization of rotations using the exponential map. *Journal of Graphics Tools 3*, 3, 29–48. ISSN 1086-7651.

HESTENES, D. 1991. The design of linear algebra and geometry. *Acta Applicandae Mathematicae 23*, 65–93.

HIGHAM, N. J. 1997. Stable iterations for the matrix square root. *Numerical Algorithms 15*, 2, 227–242.

HORN, R. A., AND JOHNSON, C. A. 1991. *Topics in Matrix Analysis*. Cambridge University press, Cambridge.

HOSCHEK, J., AND LASSER, D. 1993. Fundamentals of computer aided geometric design. ISBN 1-56881-007-5.

JOLLIFFE, I. T. 1986. *Principal Component Analysis*. Series in Statistics. Springer-Verlag.

KENNEY, C., AND LAUB, A. J. 1989. Condition estimates for matrix functions. *SIAM Journal on Matrix Analysis and Applications 10*, 2, 191–209.

KIM, M.-J., SHIN, S. Y., AND KIM, M.-S. 1995. A general construction scheme for unit quaternion curves with simple high order derivatives. *Proceedings of SIGGRAPH 95* (August), 369–376. ISBN 0-201-84776-0. Held in Los Angeles, California.

LENGYEL, J. E. 1999. Compression of time-dependent geometry. *1999 ACM Symposium on Interactive 3D Graphics* (April), 89–96. ISBN 1-58113-082-1.

MARTHINSEN, A. 2000. Interpolation in Lie groups. *SIAM Journal on Numerical Analysis 37*, 1, 269–285.

MOLER, C. B., AND LOAN, C. F. V. 1978. Nineteen dubious ways to compute the matrix exponential. *SIAM Review 20*, 801–836.

MURRAY, R. M., LI, Z., AND SASTRY, S. S. 1994. *A Mathematical Introduction to Robotic Manipulation*. CRC Press.

NAEVE, A., AND ROCKWOOD, A. 2001. Geometric algebra. SIGGRAPH 2001 course #53.

PARK, F. C., AND RAVANI, B. 1997. Smooth invariant interpolation of rotations. *ACM Transactions on Graphics 16*, 3 (July), 277–295. ISSN 0730-0301.

RAMAMOORTHI, R., AND BARR, A. H. 1997. Fast construction of accurate quaternion splines. *Proceedings of SIGGRAPH 97* (August), 287–292. ISBN 0-89791-896-7. Held in Los Angeles, California.

SHOEMAKE, K., AND DUFF, T. 1992. Matrix animation and polar decomposition. *Graphics Interface '92* (May), 258–264.

SHOEMAKE, K. 1985. Animating rotation with quaternion curves. *Computer Graphics (Proceedings of SIGGRAPH 85) 19*, 3 (July), 245–254. Held in San Francisco, California.

SHOEMAKE, K. 1991. Quaternions and 4x4 matrices. *Graphics Gems II*, 351–354. ISBN 0-12-064481-9. Held in Boston.

WEB3D CONSORTIUM. 1999. H-Anim. *http://ece.uwaterloo.ca:80/~h-anim*.

ZEFRAN, M., AND KUMAR, V. 1998. Rigid body motion interpolation. *Computer Aided Design 30*, 3, 179–189.

ZEFRAN, M., KUMAR, V., AND CROKE, C. 1996. Choice of riemannian metrics for rigid body kinematics. In *ASME 24th Biennial Mechanisms Conference*.

ZEFRAN, M. 1996. Continuous methods for motion planning. PhD-thesis, U. of Pennsylvania, Philadelphia, PA.

ZORIN, D., AND SCHRÖDER, P. 1999. Subdivision for modeling and animation. SIGGRAPH 1999 course # 47.

A Existence of matrix roots

In the following we will analyze the conditions for the existence of matrix roots, which are intuitively parts of the transformation such that all parts are identical and their combined application yields the original transformation. We will rather use this intuitive geometric point of view – a formal proof of the claims made here could be found in [Horn and Johnson 1991, Thm. 6.4.14].

First, it is clear that a reflection cannot be split into several equivalent parts and, consequently, transformation matrices must have positive determinant. This property is obviously necessary, however, not sufficient. To understand this, we need to analyze the eigenvalues of the transformation matrix as they are representative for the nature of the transformation. Note that the product of all eigenvalues is the determinant and, therefore, has to be real.

If all eigenvalues are (real) positive the transformation is a pure scale and taking roots is simple. If the eigenvalues have an imaginary part the respective transformation has a rotational (or shear) component. Because the product of all eigenvalues is real they form two conjugate groups. These groups stay conjugate when roots of the eigenvalues are taken so that determinant and, thus, the transformation matrix is still real.

A problem occurs in case of real negative eigenvalues (i.e. the imaginary part is zero), which is why we have excluded these transsformations so far. Taking roots of these values introduces imaginary parts in the determinant. Because the determinant is positive the number of negative eigenvalues has to be even, which allows one to analyze them pairwise. A pair of eigenvalues essentially defines a transformation in 2D and since both eigenvalues are real and negative this transformation contains a scale part and either a rotation by π or two reflections. If both eigenvalues have the same magnitude the transformation is a rotation by π and a uniform scale. Taking roots intuitively means reducing the angle of rotation and adjusting the uniform scale. However, if the eigenvalues have different magnitude the corresponding transformation can be seen as two reflections or as a rotation together with a non-uniform scale. It is impossible to split this transformation into equivalent parts, because the non-uniform scale is orientation dependent and the orientation changes due to the rotation. Note that compared to other rotational angles it is not possible to interpret this transformation as a shear. Rephrasing this in terms of eigenvalues: if the imaginary parts have same magnitude their roots could be assigned different signs so that they form a conjugate pair; if they have different magnitude this is not possible.

Concluding, a transformation is divisible, if the real negative eigenvalues of the matrix representing the transformation have even multiplicity. Assuming a positive determinant, it is not divisible if it has a pair of negative real eigenvalues with different magnitude. Geometrically, a pair of real negative eigenvalues with different magnitude indicate a rotation by π together with a non-uniform scale. This is the only type of transformation that cannot be handled (without further treatment) with the approach presented here. A rotation by π together with uniform scales as well as other rotational angles together with non-uniform scales are permissible. For later use we denote this class of transformation matrices $\mathbb{T}$. Note, however, that for divisible transforms with real negative eigenvalues there is no preferred choice for the primary roots and, thus, the scalar multiplication operation is not continous for such arguments.

B Matrix exponential and logarithm & Lie products

The connection between the matrix operators defined in Sections 3 and 4 and matrix exponential and logarithm is not quite obvious. Recall the definition of exponential and logarithms from Equations 3 and 4. One has to be careful when carrying over equivalence transforms for exponentials and logarithms from the scalar case, i.e. e^{A+B}, $e^A e^B$, and $e^B e^A$ are generally not the same. However, a sufficient condition for the expressions to be the same is that A and B commute (see [Horn and Johnson 1991, Thm. 6.2.38]). This leads to the identities

$$e^{mA} = e^{A+\ldots+A} = e^A \cdots e^A = \left(e^A\right)^m, \quad m \in \mathbb{N}.$$

Assuming that $e^{mA} \in \mathbb{T}$ we can take m-the roots on both sides[1],

$$\left(e^{mA}\right)^{1/m} = e^A \Leftrightarrow e^{\frac{1}{m}A} = \left(e^A\right)^{1/m}$$

thus

$$e^{rA} = \left(e^A\right)^r, \quad r \in \mathbb{Q}. \tag{9}$$

By definition $e^{\log A} = A$ and $\log e^A = A$. Setting $A = \log B$ in Eq. 9 and assuming the logarithms of both sides exist yields

$$\begin{aligned} \log\left(e^{r\log B}\right) &= \log\left(e^{\log B}\right)^r \\ r\log B &= \log\left(B^r\right). \end{aligned} \tag{10}$$

This immediately leads to the result for the scalar multiplication given in Equation 5. From this connection of roots and logarithms it is clear that real matrix logarithms exist exactly when real matrix roots exist (see also [Horn and Johnson 1991, Thm. 6.4.15]).

As said before, e^{A+B} and $e^A e^B$ are generally not the same if A and B do not commute. A way of connecting these expressions in the general case is the Lie product formula (see [Horn and Johnson 1991, Chapter 6.5] for a derivation):

$$e^{A+B} = \lim_{n\to\infty}\left(e^{\frac{1}{n}A} e^{\frac{1}{n}B}\right)^n \tag{11}$$

Applying this to $\log A, \log B$ instead of A and B leads to

$$\begin{aligned} e^{\log A + \log B} &= \lim_{n\to\infty}\left(e^{\frac{1}{n}\log A} e^{\frac{1}{n}\log B}\right)^n \\ &= \lim_{n\to\infty}\left(\left(e^{\log A}\right)^{1/n}\left(e^{\log B}\right)^{1/n}\right)^n \\ &= \lim_{n\to\infty}\left(A^{1/n}B^{1/n}\right)^n, \end{aligned} \tag{12}$$

which leads to the representation of the addition given in Equation 6. The use of the standard matrix addition in the exponent proves that the addition operator is indeed commutative.

C Implementation

The computation of exponential and logarithm of rotations or rigid body motions could be performed using Rodrigues' formula (see [Murray et al. 1994]). The transformations considered here are more general, however, including (non-uniform) scales. We are unclear whether Rodrigues' formula generalizes to this group and, therefore, propose an implementation based on matrix series. Note that Rodrigues' formula is the method of choice if scaling is not needed because it is both faster and more robust.

The computation of matrix functions such as the exponential and the logarithm is non-trivial. For example, evaluating Equation 4 for computing the exponential is numerically unstable. The preferred way of computing matrix functions is in many cases to use a Schur decomposition and evaluate the function on the upper triangle matrix [Golub and Van Loan 1989].

However, this work is intended for graphics where standard matrix packages only offer elementary matrix operations. For this reason, implementations are provided using only matrix inversion, multiplication, and addition. For the sake of completeness the pseudo-code from some of the original publications is repeated here.

Moler and van Loan [1978] have investigated several ways of computing the exponential of a matrix A in an iterative way and recommend a Padé approximation with scaling. Scaling A leads to smaller eigenvalues, which in turn, speeds up the convergence of iterative solvers. This is the pseudo-code of the procedure (see also [Golub and Van Loan 1989]):

```
Compute $X = e^A$
    $j = \max(0, 1 + \lfloor \log_2(\|A\|) \rfloor)$
    $A = 2^{-j}A$
    $D = I;\ N = I;\ X = I;\ c = 1$
    for $k = 1$ to $q$
        $c = c(q-k+1)/(k(2q-k+1))$
        $X = AX;\ N = N + cX;\ D = D + (-1)^k cX$
    end for
    $X = D^{-1}N$
    $X = X^{2^j}$
```

The number of iterations q depends on the desired accuracy, $q = 6$ has proven to be a good choice for the applications intended here.

The logarithm of a matrix A can be computed using a truncated Taylor series. However, convergence is not guaranteed or poor if A is not near the identity matrix. By using the identity $\log A = 2^k \log A^{1/2^k}$ and, thus, repeatedly taking the square root A can be made close enough to identity. Exploiting this equation and using a Taylor approximation has been introduced by Kenney and Laub [1989] and leads to the following algorithm:

```
Compute $X = \log A$
    $k = 0$
    while $\|A - I\| > 0.5$
        $A = A^{1/2}$
        $k = k + 1$
    end while
    $A = I - A$
    $Z = A;\ X = A;\ i = 1$
    while $\|Z\| > \varepsilon$
        $Z = ZA;\ i = i + 1$
        $X = X + Z/i$
    end while
    $X = 2^k X$
```

However, this algorithms needs to compute square roots of matrices. Higham [1997] has compared several iterative methods to compute matrix square roots and generally recommends the following simple method due to Denman and Beavers [1976]:

```
Compute $X = A^{1/2}$
    $X = A;\ Y = I$
    while $\|XX - A\| > \varepsilon$
        $iX = X^{-1};\ iY = Y^{-1}$
        $X = \frac{1}{2}(X + iY);\ Y = \frac{1}{2}(Y + iX)$
    end while
```

Note that all while loops in the pseudo codes should terminate after a fixed number of iterations since numerical problems might lead to poor convergence.

[1]This depends also on our choice of primary roots, which could be a problem where the primary matrix root function is discontinous, i.e. for matrices with negative real eigenvalues